

## **Measuring and Evaluating Maintenance Process Using Reliability, Risk, and Test Metrics**

**Norman F. Schneidewind, Fellow IEEE**

IEEE Transactions on Software Engineering, Vol. 25, No. 6, November/December 1999, pp. 768-781.

Computer and Information Sciences and Operations Division  
Naval Postgraduate School  
Monterey, CA 93943, U.S.A.  
Voice: (831) 656-2719  
Fax : (831) 656-3407  
Email: nschneid@nps.navy.mil

### **Abstract**

In analyzing the stability of a maintenance process, it is important that it not be treated in isolation from the reliability and risk of deploying the software that result from applying the process. Furthermore, we need to consider the efficiency of the test effort that is a part of the process and a determinate of reliability and risk of deployment. The relationship between product quality and process capability and maturity has been recognized as a major issue in software engineering based on the premise that improvements in process will lead to higher quality products. To this end, we have been investigating an important facet of process capability – stability – as defined and evaluated by trend, change, and shape metrics, across releases and within a release. Our integration of product and process measurement serves the dual purpose of using metrics to assess and predict reliability and risk and to evaluate process stability. We use the NASA Space Shuttle flight software to illustrate our approach.

**Index Terms** - Maintenance process stability, product and process integration, reliability risk.

### **INTRODUCTION**

Measuring and evaluating the stability of maintenance processes is important because of the recognized relationship between process quality and product quality [7]. We focus on the important quality factor *reliability*. A maintenance process can quickly become unstable because the very act of installing software changes the environment: pressures operate to modify the environment, the problem, and the technological solutions. Changes generated by users and the environment and the consequent need for adapting the software to the changes is unpredictable and cannot be accommodated without iteration. Programs must be adaptable to change and the resultant change process must be planned and controlled. According to Lehman, large programs are never completed, they just continue to evolve [11]. In other words, with software, we are dealing with a moving target. Maintenance is performed continuously and the stability of the maintenance process has an effect on product reliability. Therefore, when we analyzed the stability of the NASA Shuttle software maintenance process, it was important to consider the reliability of the software that the process produces. Furthermore, we needed to consider the efficiency of the test effort that is a part of the process and a determinate of reliability. Therefore, we integrated these factors into a unified model, which allowed us to measure the influence of maintenance actions and test effort on the reliability of the software. Our hypothesis was that these metrics would exhibit trends and other characteristics over time that would be indicative of the stability of the process. Our results indicate that this is the case.

We conducted research on the NASA Space Shuttle flight software to investigate a hypothesis of measuring and evaluating maintenance stability. We used several metrics and applied them across releases of the software and within releases. The trends and shapes of metric functions over time provide evidence of whether the software maintenance process is stable. We view stability as the condition of a process that results in increasing reliability, decreasing risk of deployment, and increasing test effectiveness. In addition, our focus is on process stability, not code stability. We explain our criteria for stability; describe metrics, trends, and shapes for judging stability; document the data that was collected; and show how to apply our approach. Building on our previous work of defining maintenance stability criteria and developing and applying trend metrics for stability evaluation [15], in this paper we review related research projects, introduce shape metrics for stability evaluation, apply our change metric for multiple release stability evaluation, consider the functionality of the software product in stability evaluation, and interpret the metric results in terms of process improvements.

Our emphasis in this paper is to propose a unified product and process measurement model for product evaluation and process stability analysis. The reader should focus on the model principles and not on the results obtained for the Shuttle. These are used only to illustrate the model concepts. In general, different numerical results would be obtained for other applications that use this model.

First, we review related research. Next, the concept of stability is explained and trend and shape metrics are defined. Then, we define the data and the Shuttle application environment. This is followed by an analysis of relationships among maintenance, reliability, test effort, and risk, both long term (i.e., across releases) and short term (i.e., within a release), as applied to the Shuttle. We conclude with a discussion of our attempts to relate product metrics to process improvements and to the functionality and complexity of the software.

## **RELATED RESEARCH AND PROJECTS**

A number of useful related maintenance measurement and process projects have been reported in the literature. Briand, et al, developed a process to characterize software maintenance projects [3]. They present a qualitative and inductive methodology for performing objective project characterizations to identify maintenance problems and needs. This methodology aids in determining causal links between maintenance problems and flaws in the maintenance organization and process. Although the authors' have related ineffective maintenance practices to organizational and process problems, they have not made a linkage to product reliability and process stability.

Gefen and Schneberger developed the hypothesis that maintenance proceeds in three distinct serial phases: corrective modification, similar to testing; improvement in function within the original specifications; and the addition of new applications that go beyond the original specifications [5]. Their results from a single large information system, which they studied in great depth, suggested that software maintenance is a multi-period process. In the Shuttle maintenance process, in contrast, all three types of maintenance activities are performed concurrently and are accompanied by continuous testing.

Henry, et al, found a strong correlation between errors corrected per module and the impact of the software upgrade [6]. This information can be used to rank modules by their upgrade impact during

code inspection in order to find and correct these errors before the software enters the expensive test phase. The authors treat the impact of change but do not relate this impact to process stability.

Khoshgoftarr et al used discriminant analysis in each iteration of their project to predict fault prone modules in the next iteration [10]. This approach provided an advance indication of reliability and the risk of implementing the next iteration. This study deals with product reliability but does not address the issue of process stability.

Pearse and Oman applied a maintenance metrics index to measure the maintainability of C source code before and after maintenance activities [13]. This technique allowed the project engineers to track the "health" of the code as it was being maintained. Maintainability is assessed but not in terms of process stability.

Pigoski and Nelson collected and analyzed metrics on size, trouble reports, change proposals, staffing, and trouble report and change proposal completion times [14]. A major benefit of this project was the use of trends to identify the relationship between the productivity of the maintenance organization and staffing levels. Although productivity was addressed, product reliability and process stability were not considered.

Sneed reengineered a client maintenance process to conform to the ANSI/IEEE Standard 1291, Standard for Software Maintenance [19]. This project is a good example of how a standard can provide a basic framework for a process and can be tailored to the characteristics of the project environment. Although applying a standard is an appropriate element of a good process, product reliability and process stability were not addressed.

Stark collected and analyzed metrics in the categories of customer satisfaction, cost, and schedule with the objective of focusing management's attention on improvement areas and tracking improvements over time [20]. This approach aided management in deciding whether to include changes in the current release, with possible schedule slippage, or include the changes in the next release. However, the authors did not relate these metrics to process stability.

Although there are similarities between these projects and our research, our work differs in that we integrate: 1) maintenance actions, 2) reliability, 3) test effort, and 4) risk to the safety of mission and crew of deploying the software after maintenance actions, for the purpose of analyzing and evaluating the stability of the maintenance process.

## **CONCEPT OF STABILITY**

### **TREND METRICS**

To gain insight about the interaction of the maintenance process with product metrics like reliability, two types of metrics are analyzed: trend and shape. Both types are used to assess and predict maintenance process stability across (long-term) and within (short-term) releases after the software is released and maintained. Shape metrics are described in the next section. By chronologically ordering metric values by release date, we obtain discrete functions in time that can be analyzed for trends across releases. Similarly, by observing the sequence of metric values as continuous functions of increasing test time, we can analyze trends within releases. These metrics are

defined as empirical and predicted functions that are assigned values based on release date (long term) or test time (short term). When analyzing trends, we note whether an increasing or decreasing trend is favorable [15]. For example, an *increasing* trend in Time to Next Failure and a *decreasing* trend in Failures per KLOC would be favorable. Conversely, a *decreasing* trend in Time to Next Failure and an *increasing* trend in Failures per KLOC would be unfavorable. A favorable trend is indicative of maintenance stability if the functionality of the software has increased with time across releases and within releases. Increasing functionality is the norm in software projects due to the enhancement that users demand over time. We impose this condition because if favorable trends are observed, they could be the result of decreasing functionality rather than having achieved maintenance stability. When trends in these metrics over time are favorable (e.g., increasing reliability), we conclude that the maintenance process is *stable* with respect to the software metric (reliability). Conversely, when the trends are unfavorable (e.g., decreasing reliability), we conclude that process is *unstable*. Our research investigated whether there were relationships among the following factors: 1) maintenance actions, 2) reliability, and 3) test-effort. We use the following types of trend metrics:

1. Maintenance actions: KLOC Change to the Code (i.e., amount of code changed necessary to add given functionality);
2. Reliability: Various reliability metrics (e.g., MTTF, Total Failures, Remaining Failures, and Time to Next Failure); and
3. Test effort: Total Test Time.

### Change Metric

Although looking for a trend on a graph is useful, it is not a precise way of measuring stability, particularly if the graph has peaks and valleys and the measurements are made at discrete points in time. Therefore, we developed a Change Metric (CM), which is computed as follows:

1. Note the change in a metric from one release to the next (i.e., release  $j$  to release  $j+1$ ).
- 2.a. If the change is in the desirable direction (e.g., Failures/KLOC decrease), treat the change in 1 as positive.
- b. If the change is in the undesirable direction (e.g., Failures/KLOC increase), treat the change in 1 as negative.
3. a. If the change in 1 is an increase, divide it by the value of the metric in release  $j+1$ .
- b. If the change in 1 is a decrease, divide it by the value of the metric in release  $j$ .
4. Compute the average of the values obtained in 3, taking into account sign. This is the change metric (CM). The CM is a quantity in the range  $-1, 1$ . A positive value indicates stability; a negative value indicates instability. The numeric value of CM indicates the degree of stability or instability. For example, .1 would indicate marginal stability and .9 would indicate high stability. Similarly, -. 1 would indicate marginal instability and -. 9 would indicate high instability. The standard deviation of these values can also be computed. Note that CM only pertains to stability or instability *with respect to the particular metric that has been evaluated* (e.g., Failures/KLOC). The evaluation of stability should be

made with respect to a set of metrics and not a single metric. The average of the CM for a set of metrics can be computed to obtain an overall metric of stability.

## **SHAPE METRICS**

In addition to trends in metrics, the shapes of metric functions provide indicators of maintenance stability. We use shape metrics to analyze the stability of an individual release and the trend of these metrics across releases to analyze long-term stability. The rationale of these metrics is that it is better to reach important points in the growth of product reliability sooner than later. If we reach these points late in testing, it is indicative of a process that is late in achieving stability. We use the following types of shape metrics:

1. Direction and magnitude of the slope of a metric function (e.g., failure rate decreases asymptotically with total test time). Using failure rate as an example within a release, it is desirable that it rapidly decrease towards zero with increasing total test time and that it have small values.
2. Percent of total test time at which a metric function changes from unstable (e.g., increasing failure rate) to stable (e.g., decreasing failure rate) and remains stable. Across releases, it is desirable that the total test time at which a metric function becomes stable gets progressively smaller.
3. Percent of total test time at which a metric function increases at a maximum rate in a favorable direction (e.g., failure rate has maximum negative rate of change). Using failure rate as an example, it is desirable for it to achieve maximum rate of decrease as soon as possible, as a function of total test time.
4. Test time at which a metric function reaches its maximum value (e.g., test time at which failure rate reaches its maximum value). Using failure rate as an example, it is desirable for it to reach its maximum value (i.e., transition from unstable to stable) as soon as possible, as a function of total test time.
5. Risk: Probability of *not* meeting reliability and safety goals (e.g., time to next failure should exceed mission duration), using various shape metrics as indicators of risk. Risk would be low if the conditions in 1-4 above obtain.

## **METRICS FOR LONG-TERM ANALYSIS**

We use certain metrics only for long-term analysis. As an example, we compute the following trend metrics over a sequence of releases:

1. Mean Time to Failure (MTTF).
2. Total Failures normalized by KLOC Change to the Code.
3. Total Test Time normalized by KLOC Change to the Code.
4. Remaining Failures normalized by KLOC Change to the Code.

## 5. Time to Next Failure.

### **METRICS FOR LONG-TERM AND SHORT-TERM ANALYSIS**

We use other metrics for both long-term and short-term analysis. As an example, we compute the following trend (1) and shape (2, 3, 4, and 5) metrics over a sequence of releases and within a given release:

1. Percent of Total Test Time required for Remaining Failures to reach a specified value.
2. Degree to which Failure Rate asymptotically approaches zero with increasing Total Test Time.
3. Percent of Total Test Time required for Failure Rate to become stable and remain stable.
4. Percent of Total Test Time required for Failure Rate to reach maximum decreasing rate of change (i.e., slope of the failure rate curve).
5. Maximum Failure Rate and Total Test Time where Failure Rate is maximum.

### **DATA AND EXAMPLE APPLICATION**

We use the Shuttle application to illustrate the concepts. This large maintenance project has been evolving with increasing functionality since 1983 [2]. We use data collected from the developer of the flight software of the NASA Space Shuttle, as shown in Table 1, which has two parts: 1 and 2. This table shows Operational Increments (OIs) of the Shuttle: OIA... OIQ, covering the period 1983-1997. We define an OI as follows: a software system comprised of modules and configured from a series of builds to meet Shuttle mission functional requirements [16]. In Part 1, for each of the OIs, we show the Release Date (the date of release by the contractor to NASA), Total Post Delivery Failures, and Failure Severity (decreasing in severity from "1" to "4"). In Part 2, we show the maintenance change to the code in KLOC (source language changes and additions) and the total test time of the OI. In addition, for those OIs with at least two failures, we show the computation of MTTF, Failures/KLOC, and Total Test Time/KLOC. KLOC is an indicator of maintenance actions, not functionality [8]. Increased functionality, as measured by the increase in the size of principal functions loaded into mass memory, has averaged about 2% over the last 10 OIs. Therefore, if a stable process were observed, it could not be attributed to decreasing functionality. Also to be noted is that the software developer is a CMM Level 5 organization that has continually improved its process.

Because the flight software is run continuously, around the clock, in simulation, test, or flight, Total Test Time refers to continuous execution time from the time of release. For OIs where there was a sufficient sample size (i.e., Total Post Delivery Failures) -- OIA, OIB, OIC, OID, OIE, OIJ, and OIO -- we predicted software reliability. For these OIs, we show Launch Date, Mission Duration, and Reliability Prediction date (i.e., the date when we made a prediction). Fortunately, for the safety of the crew and mission, there have been few post delivery failures. Unfortunately, from the standpoint of prediction, there is a sparse set of observed failures from which to estimate reliability model parameters, particularly for recent OIs. Nevertheless, we predict reliability prior to launch date for OIs with as few as five failures spanning many months of maintenance and testing. In the case of OIE, we

predict reliability after launch because no failures had occurred prior to launch to use in the prediction model. Because of the scarcity of failure data, we made predictions using all severity levels of failure data. This turns out to be beneficial when making reliability risk assessments using number of Remaining Failures. For example, rather than specifying that the number of predicted Remaining Failures must not exceed one severity “1”, the criterion could specify that the prediction not exceed one failure of *any type* – a more conservative criterion [16].

As would be expected, the number of pre-delivery failures is much greater than the number of post delivery failures because the software is not as mature from a reliability standpoint. Thus, a way around the insufficient sample size of recent OIs for reliability prediction is to use pre-delivery failures for model fit and then use the fitted model to predict post-delivery failures. However, we are not sure that this approach is appropriate because the multiple builds in which failures can occur and the test strategies used to attempt to crash various pieces of code during the pre-delivery process contrast sharply with the post-delivery environment of testing an integrated OI with operational scenarios. Nevertheless, we are experimenting with this approach in order to evaluate the prediction accuracy. The results will be reported in a future paper.

Table 1-Part 1: Characteristics of Maintained Software Across <i>Shuttle</i> Releases						
Operational Increment	Release Date	Launch Date	Mission Duration (Days)	Reliability Prediction Date	Total Post Delivery Failures	Failure Severity
A	9/1/83	No Flights		12/9/85	6	One 2 Five 3
B	12/12/83	8/30/84	6	8/14/84	10	Two 2 Eight 3
C	6/8/84	4/12/85	7	1/17/85	10	Two 2 Seven 3 One 4
D	10/5/84	11/26/85	7	10/22/85	12	Five 2 Seven 3
E	2/15/85	1/12/86	6	5/11/89	5	One 2 Four 3
F	12/17/85				2	Two 3
G	6/5/87				3	One 1 Two 3
H	10/13/88				3	Two 1 One 3
I	6/29/89				3	Three 3
J	6/18/90	8/2/91	9	7/19/91	7	Seven 3
K	5/2/91				1	One 1
L	6/15/92				3	One 1 One 2 One 3
M	7/15/93				1	One 3
N	7/13/94				1	One 3
O	10/18/95	11/19/96	18	9/26/96	5	One 2 Four 3
P	7/16/96				3	One 2 Two 3
Q	3/5/97				1	One 3

Table 1-Part 2: Characteristics of Maintained Software Across <i>Shuttle</i> Releases					
Operational Increment	KLOC Change	Total Test Time (Days)	MTTF (Days)	Total Failures/KLOC Change	Total Test Time/ KLOC Change (Days)
A	8.0	1078	179.7	0.750	134.8
B	11.4	4096	409.6	0.877	359.3
C	5.9	4060	406.0	1.695	688.1
D	12.2	2307	192.3	0.984	189.1
E	8.8	1873	374.6	0.568	212.8
F	6.6	412	206.0	0.303	62.4
G	6.3	3077	1025.7	0.476	488.4
H	7.0	540	180.0	0.429	77.1
I	12.1	2632	877.3	0.248	217.5
J	29.4	515	73.6	0.238	17.5
K	21.3	182			8.5
L	34.4	1337	445.7	0.087	38.9
M	24.0	386			16.1
N	10.4	121			11.6
O	15.3	344	68.8	0.327	22.5
P	7.3	272	90.7	0.411	37.3
Q	11.0	75			6.8

## **RELATIONSHIPS AMONG MAINTENANCE, RELIABILITY, AND TEST EFFORT**

### **METRICS FOR LONG-TERM ANALYSIS**

We want our maintenance effort to result in **increasing** reliability of software over a sequence of releases. A graph of this relationship over calendar time and the accompanying CM calculations indicate whether the long-term maintenance effort has been successful as it relates to reliability. In order to measure whether this is the case, we use both predicted and actual values of metrics. We predict reliability in advance of deploying the software. If the predictions are favorable, we have confidence that the risk is

acceptable to deploy the software. If the predictions are unfavorable, we may decide to delay deployment and perform additional inspection and testing. Another reason for making predictions is to assess whether the maintenance process is effective in improving reliability and to do it sufficiently early during maintenance to improve the maintenance process. In addition to making predictions, we collected and analyzed historical reliability data. These data show in retrospect whether maintenance actions were successful in increasing reliability. In addition, the test effort should not be disproportionate to the amount of code that is changed and to the reliability that is achieved as a result of maintenance actions.

### Mean Time to Failure

We want Mean Time to Failure (MTTF), as computed by equation (1), to show an **increasing** trend across releases, indicating **increasing** reliability.

$$\text{Mean Time to Failure} = \text{Total Test Time} / \text{Total Number of Failures During Test} \quad (1)$$

### Total Failures

Similarly, we want Total Failures (and faults), normalized by KLOC Change in Code, as computed by equation (2), to show a **decreasing** trend across releases, indicating that reliability is **increasing** with respect to code changes.

$$\text{Total Failures/KLOC} = \text{Total Number of Failures During Test} / \text{KLOC Change in Code on the OI} \quad (2)$$

We plot Equations (1) and (2) in Figure 1 and Figure 2, respectively, against Release Time of OI. This is the number of months since the release of the OI, using "0" as the release time of OIA. We identify the OIs at the bottom of the plots. Both of these plots use actual values (i.e., historical data). The CM value for equation (1) is **-0.060** indicating small instability with respect to MTTF and **0.087** for equation (2) indicating small stability with respect to normalized Total Failures. The corresponding standard deviations are 0.541 and 0.442. Large variability in CM is the case in this application due to the large variability in functionality across releases. Furthermore, it is not our objective to judge the process that is used in this example. Rather, our purpose in showing these and subsequent values of CM is to *illustrate* our model. We use these plots and the CM to assess the long-term stability of the maintenance process. We show example computations of CM for equations (1) and (2) in Table 2.

Table 2: Example Computations of Change Metric (CM)				
Operational Increment	MTTF (Days)	Relative Change	Total Failures/KLOC	Relative Change
A	179.7		0.750	
B	409.6	0.562	0.877	-0.145
C	406.0	-0.007	1.695	-0.483
D	192.3	-0.527	0.984	0.419
E	374.6	0.487	0.568	0.423
J	73.6	-0.805	0.238	0.581
O	68.8	-0.068	0.330	-0.272
	<b>CM</b>	<b>-0.060</b>	<b>CM</b>	<b>0.087</b>

## Total Test Time

We want Total Test Time, normalized by KLOC Change in Code, as computed by equation (3), to show a **decreasing** trend across releases, indicating that test effort is **decreasing** with respect to code changes.

$$\text{Total Test Time/KLOC} = \text{Total Test Time/KLOC Change in Code on the OI} \quad (3)$$

We plot Equation (3) in Figure 3 against Release Time of OI, using actual values. The CM value for this plot is **0.116**, with a standard deviation of 0.626, indicating stability with respect to efficiency of test effort. We use this plot and the CM to assess whether testing is efficient with respect to the amount of code that has been changed.

## Reliability Predictions

### Total Failures

Up to this point, we have used only actual data in the analysis. Now we expand the analysis to use both predictions and actual data but only for the seven OIs where we could make predictions. Using the Schneidewind Model [1], [9], [16], [17], [18] and the SMERFS software reliability tool [4], we show prediction equations, using 30 day time intervals, and make predictions for OIA, OIB, OIC, OID, OIE, OIJ, and OIO. This model or any other applicable model may be used [1], [4].

To predict Total Failures in the range [1,4] (i.e., failures over the life of the software), we use equation (4):

$$F(\infty) = \alpha / \beta + X_{s-1} \quad (4)$$

where the terms are defined as follows:

s: starting time interval for using failures counts for computing parameters  $\alpha$  and  $\beta$ ,

$\alpha$ : initial failure rate,

$\beta$ : rate of change of failure rate, and

$X_{s-1}$ : observed failure count in the range [1,s-1].

Now, we predict Total Failures normalized by KLOC Change in Code. We want predicted normalized Total Failures to show a **decreasing** trend across releases. We computed a CM value for this data of **.115**, with a standard deviation of .271, indicating stability with respect to predicted normalized Total Failures.

### Remaining Failures

To predict Remaining Failures  $r(t)$  at time  $t$ , we use equation (5) [1], [9], [17]:

$$r(t) = F(\infty) - X_t \quad (5)$$

This is the predicted Total Failures over the life of the software minus the observed failure count at time  $t$ .

We predict Remaining Failures, normalize them by KLOC Change in Code, and compare them with normalized actual Remaining Failures for seven OIs in Figure 4. We approximate Actual Remaining Failures at time  $t$  by subtracting the observed failure count at time  $t$  from the observed Total Failure count at time  $T$ , where  $T \gg t$ . The reason for this approach is that we are approximating the failure count over the life of the software by using the failure count at time  $T$ . We want equation (5) and actual Remaining Failures, normalized by KLOC Change in Code, to show a **decreasing** trend over a sequence of releases. The CM values for these plots are **0.107** and **0.277**, respectively, indicating stability with respect to Remaining Failures. The corresponding standard deviations are .617 and .715.

### Time to Next Failure

To predict the Time for the Next  $F_t$  Failures to occur, when the current time is  $t$ , we use equation (6)

$$T_F(t) = [(\log[\alpha / (\alpha - \beta(X_{s,t} + F_t))]) / \beta] - (t - s + 1) \quad (6)$$

[1], [16], [17].

The terms in  $T_F(t)$  have the following definitions:

$t$ : Current time interval;

$X_{s,t}$ : Observed failure count in the range  $[s,t]$ ; and

$F_t$ : Given number of failures to occur after interval  $t$  (e.g., one failure).

We want equation (6) to show an **increasing** trend over a sequence of releases. Predicted and actual values are plotted for six OIs (OIO has no failures) in Figure 5. The CM values for these plots are **-0.152** and **-0.065**, respectively, indicating slight instability with respect to Time to Next Failure. The corresponding standard deviations are .693 and .630.

We predicted values of Total Failures, Remaining Failures, and Time to Next Failure as indicators of the risk of operating software in the future: is the predicted future reliability of software an acceptable risk? The risk to the mission may or may not be acceptable. If the latter, we take action to improve the maintained product or the maintenance process. We use actual values to measure the reliability of software and the risk of deploying it resulting from maintenance actions.

### Summary

We summarize change metric values in Table 3. Overall (i.e., average CM), the values indicate marginal stability. If the majority of the results and the average CM were negative, this would be an alert to investigate the cause. The results could be caused by: 1) greater functionality and complexity in the software over a sequence of releases, 2) a maintenance process that needs to be improved, or 3) a combination of these causes.

Table 3: Change Metric Summary		
Metric	Actual	Predicted
Mean Time To Failure	-0.060	
Total Test Time per KLOC	0.116	
Total Failures per KLOC	0.087	0.115
Remaining Failures per KLOC	0.277	0.107
Time to Next Failure	-0.065	-0.152
<b>Average</b>	<b>0.071</b>	

### **METRICS FOR LONG-TERM AND SHORT-TERM ANALYSIS**

In addition to the long-term maintenance criteria, it is desirable that the maintenance effort results in increasing reliability within each release or OI. One way to evaluate how well we achieve this goal is to predict and observe the amount of test time that is required to reach a specified number of Remaining Failures. In addition, we want the test effort to be efficient in finding residual faults for a given OI. Furthermore, number of Remaining Failures serves as an indicator of the risk involved in using the maintained software (i.e., a high value of Remaining Failures portends a significant number of residual faults in the code). In the analysis that follows we use predictions and actual data for a selected OI to *illustrate* the process: OID.

#### **Total Test Time Required for Specified Remaining Failures**

We predict the Total Test Time that is required to achieve a specified number of Remaining Failures,  $r(t_t)$ , at time  $t_t$ , by equation (7) [1], [17]:

$$t_t = [\log[\alpha / (\beta[r(t_t)])]] / \beta + (s - 1) \quad (7)$$

We plot predicted and actual Total Test Time for OID in Figure 6 against given number of Remaining Failures. The two plots have similar shapes and show the typical asymptotic characteristic of reliability (e.g., Remaining Failures) versus Total Test Time. These plots indicate the possibility of big gains in reliability in the early part of testing; eventually the gains become marginal as testing continues. The figure also shows how risk is reduced with a decrease in Remaining Failures that is accomplished with increased testing. Predicted values are used to gauge how much maintenance test effort would be required to achieve desired reliability goals and whether the predicted amount of Total Test Time is technically and economically feasible. We use actual values to judge whether the maintenance test effort has been efficient in relation to the achieved reliability.

#### **Failure Rate**

In the short-term (i.e., within a release), we want the Failure Rate (1/MTTF) of an OI to **decrease** over an OI's Total Test Time, indicating **increasing** reliability. Practically, we would look for a decreasing trend, after an initial period of instability (i.e., increasing rate as personnel learn how to maintain new software). In addition, we use various shape metrics, as defined previously, to see how quickly we can achieve reliability

growth with respect to test time expended. Furthermore, Failure Rate is an indicator of the risk involved in using the maintained software (i.e., an increasing failure rate indicates an increasing probability of failure with increasing use of the software).

$$\text{Failure Rate} = \text{Total Number of Failures During Test} / \text{Total Test Time} \quad (8)$$

We plot Equation (8) for OID in Figure 7 against Total Test Time since the release of OID. Figure 7 *does* show that short-term stability is achieved (i.e., failure rate asymptotically approaches zero with increasing Total Test Time). In addition, this curve shows when the failure rate transitions from unstable (positive Failure Rate) to stable (negative Failure Rate). The figure also shows how risk is reduced with decreasing Failure Rate as the maintenance process stabilizes. Furthermore, in Figure 8 we plot the rate of change (i.e., slope) of the Failure Rate of Figure 7. This curve shows the percent of Total Test Time when the rate of change of Failure Rate reaches its maximum negative value. We use these plots to assess whether we have achieved short-term stability in the maintenance process (i.e., whether Failure Rate decreases asymptotically with increasing Total Test Time). If we obtain contrary results, this would be an alert to investigate whether this is caused by: 1) greater functionality and complexity of the OI as it is being maintained, 2) a maintenance process that needs to be improved, or 3) a combination of these causes.

Another way of looking at failure rate with respect to stability and risk is the annotated Failure Rate of OID shown in Figure 9, where we show both the actual and predicted Failure Rates. We use equations (8) and (9) [1] to compute the actual and predicted Failure Rates, respectively, where  $i$  is a vector of time intervals for  $i \geq s$  in equation (9).

$$f(i) = \alpha(\text{EXP}(-\beta(i-s+1))) \quad (9)$$

A 30-day interval has been found to be convenient as a unit of Shuttle test time because testing can last for many months or even years. Thus this is the unit used in Figure 9, where we show the following events in intervals, where the predictions were made at 12.73 intervals:

Release time: 0 interval,  
 Launch time: 13.90 intervals,  
 Predicted time of maximum Failure Rate: 6.0 intervals,  
 Actual time of maximum Failure Rate: 7.43 intervals,  
 Predicted maximum Failure Rate: .5735 failures per interval, and  
 Actual maximum Failure Rate: .5381 failures per interval.

In Figure 9, stability is achieved after the maximum failure rate occurs. This is at  $i = s$  (i.e.  $i = 6$  intervals) for predictions because equation (9) assumes a monotonically decreasing failure rate, whereas the actual failure rate increases, reaches a maximum at 7.43 intervals, and then decreases. Once stability is achieved, risk decreases.

## **Summary**

In addition to analyzing short-term stability with these metrics, we use them to analyze long-term stability across releases. We show the results in Table 4 where the percent of Total Test Time to achieve reliability growth goals is tabulated for a set of OIs, using actual failure data, and the Change Metrics are computed. Overall, the values of CM indicate marginal instability. Interestingly, except for OID, the

maximum negative rate of change of failure rate occurs when Failure Rate becomes stable, suggesting that maximum reliability growth occurs when the maintenance process stabilizes.

Table 4: Percent of Total Test Time Required to Achieve Reliability Goals and Change Metrics (CM)						
Operational Increment	One Remaining Failure (% Test Time)	Relative Change	Stable Failure Rate (% Test Time)	Relative Change	Maximum Failure Rate Change (% Test Time)	Relative Change
A	77.01		76.99		76.99	
B	64.11	0.168	64.11	0.167	64.11	0.167
C	32.36	0.495	10.07	0.843	10.07	0.843
D	84.56	-0.617	12.70	-0.207	22.76	-0.558
E	83.29	0.015	61.45	-0.793	61.45	-0.630
J	76.88	0.077	76.89	-0.201	76.89	-0.201
O	46.49	0.395	100.00	-0.231	100.00	-0.231
	<b>CM</b>	<b>0.089</b>	<b>CM</b>	<b>-0.070</b>	<b>CM</b>	<b>-0.101</b>
	<b>STD DEV</b>	<b>0.392</b>	<b>STD DEV</b>	<b>0.543</b>	<b>STD DEV</b>	<b>0.544</b>

### **SHUTTLE OPERATIONAL INCREMENT FUNCTIONALITY AND PROCESS IMPROVEMENT**

Table 5 shows the major functions of each OI [12] along with the Release Date and KLOC Change repeated from Table 1. There is not a one-for-one relationship between KLOC Change and the functionality of the change because, as stated earlier, KLOC is an indicator of maintenance actions, not functionality. However, the software developer states that there has been increasing software functionality and complexity with each OI, in some cases with *less* rather than more KLOC [8]. The focus of the early OIs was on launch, orbit, and landing. Later OIs, as indicated in Table 5, built upon this baseline functionality to add greater functionality in the form of MIR docking and the Global Positional System, for example. Table 6 shows the process improvements that have been made over time on this project, indicating continuous process improvement across releases.

The stability analysis that was performed yielded mixed results: about half are favorable and half are unfavorable. Some variability in the results may be due to gaps in the data caused by OIs that have experienced insufficient failures to permit statistical analysis. Also, we note that the values of CM are marginal for both the favorable and unfavorable cases. Although there is not pronounced stability neither is there pronounced instability. If there were consistent and large negative values of CM, it would be cause for alarm and would suggest the need to perform a thorough review of the process. This is not the case for the Shuttle. We suspect but cannot prove that in the absence of the process improvements of Table 6, the CM values would look much worse. It is very difficult to associate a specific product improvement with a specific process improvement. A controlled experiment would be necessary to hold all process factors constant and observe the one factor of interest and its influence on product quality. This is infeasible to do in industrial organizations. However, we suggest that in the *aggregate* a series of process improvements is beneficial for product quality and that a *set* of CM values can serve to highlight possible process problems.

Table 5: Shuttle Operational Increment Functionality			
Operational Increment	Release Date	KLOC Change	Operational Increment Function
A	9/1/83	8.0	Redesign of Main Engine Controller.
B	12/12/83	11.4	Payload Re-manifest Capabilities.
C	6/8/84	5.9	Crew Enhancements.
D	10/5/84	12.2	Experimental Orbit Autopilot. Enhanced Ground Checkout.
E	2/15/85	8.8	Western Test Range. Enhance Propellant Dumps.
F	12/17/85	6.6	Centaur.
G	6/5/87	6.3	Post 51-L (Challenger) Safety Changes.
H	10/13/88	7.0	System Improvements.
I	6/29/89	12.1	Abort Enhancements.
J	6/18/90	29.4	Extended Landing Sites. Trans-Atlantic Abort Code Co-Residency.
K	5/21/91	21.3	Redesigned Abort Sequencer. One Engine Auto Contingency Aborts. Hardware Changes for New Orbiter.
L	6/15/92	34.4	Abort Enhancements.
M	7/15/93	24.0	On-Orbit Changes.
N	7/13/94	10.4	MIR Docking. On-Orbit Digital Autopilot Changes.
O	10/18/95	15.3	Three Engine Out Auto Contingency.
P	7/16/96	7.3	Performance Enhancements.
Q	3/5/97	11.0	Single Global Positioning System.

Table 6: Chronology of Process Improvements	
Year in which Process Improvement Introduced	Process Improvement
1976	Structured Flows
1977	Formal Software Inspections
1978	Formal Inspection Moderators
1980	Formalized Configuration Control
1981	Inspection Improvements
1982	Configuration Management Database
1983	Oversight Analyses
1984	Build Automation Formalized Requirements Analysis
1985	Quarterly Quality Reviews Prototyping
1986	Inspection Improvements Formal Requirements Inspections
1987	Process Applied to Support Software
1988	Reconfiguration Certification Reliability Modeling and Prediction
1989	Process Maturity Measurements
1990	Formalized Training
1992	Software Metrics

## CONCLUSIONS

As stated in the Introduction, our emphasis in this paper was to propose a unified product and process measurement model for both product evaluation and process stability analysis. We were less interested in the results of the Shuttle stability analysis, which were used to illustrate the model concepts. We conclude, based on both predictive and retrospective use of reliability, risk, and test metrics, that it is feasible to measure and assess both product quality and the stability of a maintenance process. The model is not domain specific. Different organizations may obtain different numerical results and trends than the ones we obtained for the Shuttle.

## Acknowledgments

We acknowledge the support provided for this project by Dr. William Farr, Naval Surface Warfare Center; Mr. Ted Keller of IBM; and Ms. Patti Thornton and Ms. Julie Barnard of United Space Alliance. We also wish to thank the anonymous reviewers for their helpful comments.

## References

- [1] Recommended Practice for Software Reliability, R-013-1992, American National Standards Institute/American Institute of Aeronautics and Astronautics, 370 L'Enfant Promenade, SW, Washington, DC 20024, 1993.
- [2] C. Billings, et al, "Journey to a Mature Software Process", IBM Systems Journal, Vol. 33, No. 1, 1994, pp. 46-61.
- [3] Lionel C. Briand, Victor R. Basili, and Yong-Mi Kim, "Change Analysis Process to Characterize Software Maintenance Projects", Proceedings of the International Conference on Software Maintenance, Victoria, British Columbia, Canada, September 19-23, 1994, pp. 38-49.
- [4] William H. Farr and Oliver D. Smith, Statistical Modeling and Estimation of Reliability Functions for Software (SMERFS) Users Guide, NAVSWC TR-84-373, Revision 3, Naval Surface Weapons Center, Revised September 1993.
- [5] David Gefen and Scott L. Schneberger, "The Non-Homogeneous Maintenance Periods: A Case Study of Software Modifications", Proceedings of the International Conference on Software Maintenance, Monterey, California, November 4-8, 1996, pp. 134-141.
- [6] Joel Henry, Sallie Henry, Dennis Kafura, and Lance Matheson, "Improving Software Maintenance at Martin Marietta", IEEE Software, Vol. 11, No.4, July 1994, pp. 67-75.
- [7] Craig Hollenbach, et al, "Combining Quality and Software Improvement", Communications of the ACM, Vol. 40, No.6, June 1997, pp. 41-45.
- [8] Private communication with Ted Keller, IBM, April 1998.
- [9] Ted Keller, Norman F. Schneidewind, and Patti A. Thornton "Predictions for Increasing Confidence in the Reliability of the Space Shuttle Flight Software", Proceedings of the AIAA Computing in Aerospace 10, San Antonio, TX, March 28, 1995, pp. 1-8.
- [10] Taghi M. Khoshgoftaar, Edward B. Allen, Robert Halstead, and Gary P. Trio, "Detection of Fault-Prone Software Modules During a Spiral Life Cycle", Proceedings of the International Conference on Software Maintenance, Monterey, California, November 4-8, 1996, pp. 69-76.
- [11] Meir M. Lehman, "Programs, Life Cycles, and Laws of Software Evolution", Proceedings of the IEEE, Vol. 68, No. 9, September 1980.
- [12] "Software Release Schedules", Lockheed Martin, January 30, 1998.

- [13] Troy Pearse and Paul Oman, "Maintainability Measurements on Industrial Source Code Maintenance Activities", Proceedings of the International Conference on Software Maintenance, Opio (Nice), France, October 17-20, 1995, pp. 295-303.
- [14] Thomas M. Pigoski and Lauren E. Nelson, "Software Maintenance Metrics: A Case Study", Proceedings of the International Conference on Software Maintenance, Victoria, British Columbia, Canada, September 19-23, 1994, pp. 392-401.
- [15] Norman F. Schneidewind, "Measuring and Evaluating Maintenance Process Using Reliability, Risk, and Test Metrics", Proceedings of the International Conference on Software Maintenance, Bari, Italy, October 2, 1997, pp. 232-239.
- [16] Norman F. Schneidewind, "Reliability Modeling for Safety Critical Software", IEEE Transactions on Reliability, Vol. 46, No.1, March 1997, pp.88-98.
- [17] Norman F. Schneidewind, "Software Reliability Model with Optimal Selection of Failure Data", IEEE Transactions on Software Engineering, Vol. 19, No. 11, November 1993, pp. 1095-1104.
- [18] Norman F. Schneidewind and T. W. Keller, "Application of Reliability Models to the Space Shuttle", IEEE Software, Vol. 9, No. 4, July 1992 pp. 28-33.
- [19] Harry Sneed, "Modelling the Maintenance Process at Zurich Life Insurance", Proceedings of the International Conference on Software Maintenance, Monterey, California, November 4-8, 1996, pp. 217-226.
- [20] George E. Stark, "Measurements for Managing Software Maintenance", Proceedings of the International Conference on Software Maintenance, Monterey, California, November 4-8, 1996, pp. 152-161.

Dr. Norman F. Schneidewind is Professor of Information Sciences and Director of the Software Metrics Research Center at the Naval Postgraduate School. He is the developer of the Schneidewind software reliability model that has been used by NASA to assist in the prediction of software reliability of the Space Shuttle -- one of the models recommended by the American National Standards Institute and the American Institute of Aeronautics and Astronautics Recommended Practice for Software Reliability. Dr. Schneidewind is a Fellow of the IEEE, elected for "contributions to software measurement models in reliability and metrics, and for leadership in advancing the field of software maintenance". In 1992 and 1998 he received an award for outstanding research achievements by the Naval Postgraduate School. In 1993 he received the IEEE Computer Society's Outstanding Contribution Award "for work leading to the establishment of IEEE Standard 1061-1992". In addition, he received the IEEE Computer Society Meritorious Service Award "for his long-term committed work in advancing the cause of software engineering standards". He was recognized for his contributions to the IEEE Computer Society by being named to the "Golden Core" of volunteers.

Figure 1. Mean Time To Failure Across Releases

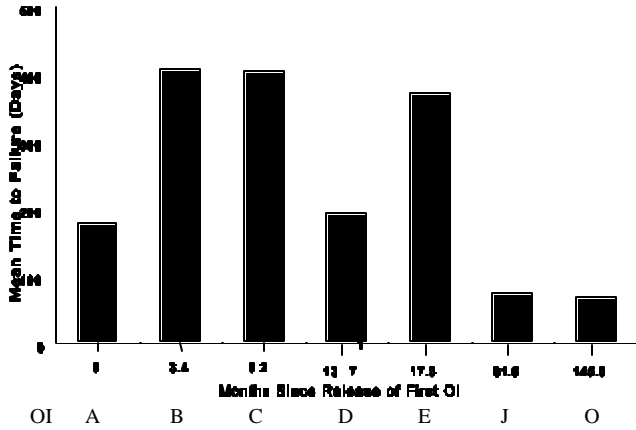


Figure 2. Total Failures per KLOC Across Releases

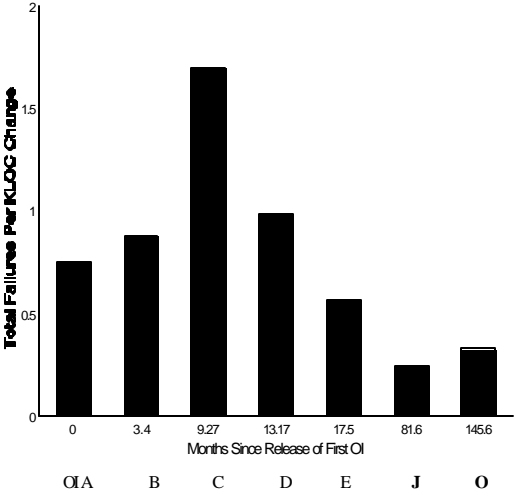


Figure 3. Total Test Time per KLOC Across Releases

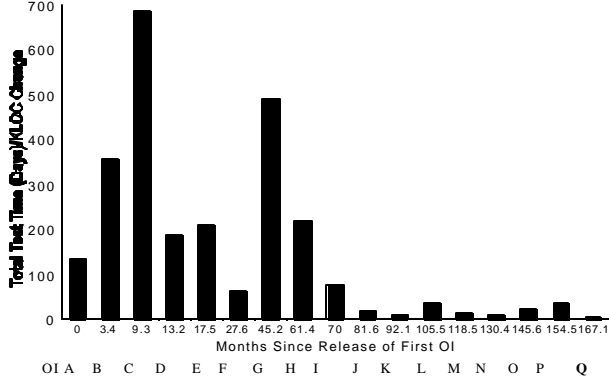
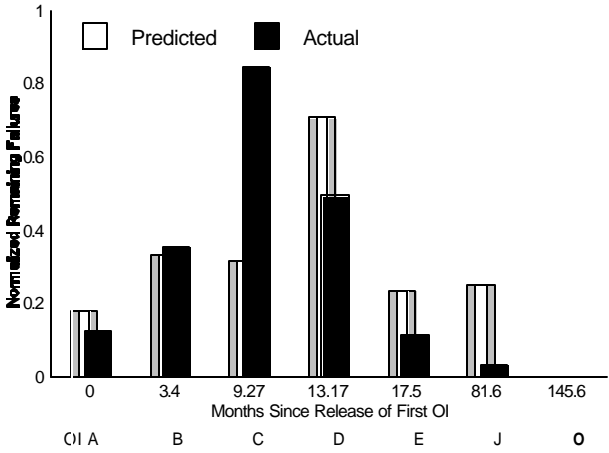


Figure 4. Reliability of Maintained Software -- Remaining Failures Normalized by Change to Code



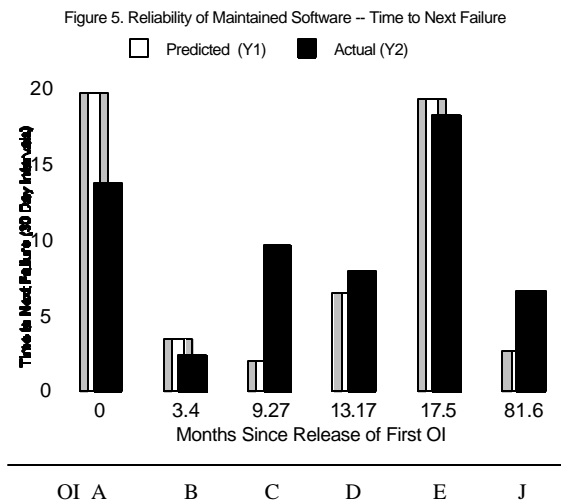


Figure 6. Total Test Time to Achieve Remaining Failures

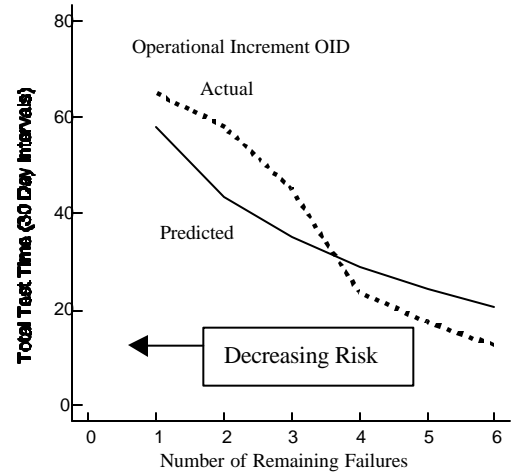


Figure 7. OID Failure Rate

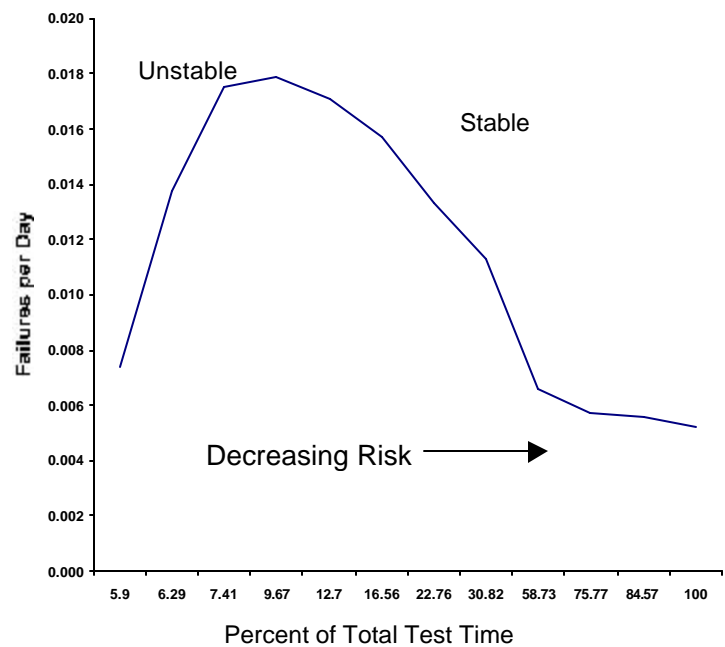


Figure 8. OID Rate of Change of Failure Rate

